IN THE SPECIFICATION:

Page 8, line 13 to page 9, line 19, please replace the existing paragraph with the following paragraph:

Functional software components of a system 20 that incorporates aspects of the present invention are shown in Fig. 1. The system 20 has a front end portion 22 that receives and compiles information, such as information from a native source file 24. The native source file 24 may be any type of source language file that is used for software development. The front end 22 produces a common language file 26 that can be fed back into the front end system 22 and/or executed by an execution environment, such as execution environment 32. As illustrated in Figure 1, the output of front end 22 may include metadata 28 and executable instructions 30, wherein the executable instructions 30 may also be referred to as "instruction code." Executable instructions 30 can be either instructions that can be directly executed by a processor (e.g. object or native machine code) or an "intermediate" type instruction (e.g. Java bytecodes, p-code, or other intermediate language) that is executed within some type of execution environment. In one embodiment, executable instructions 30 comprise a "common" (in the sense of universal) intermediate language suitable for representing the concepts of a plurality of different types of source languages, so that only one type of intermediate language need be used regardless of the specific source language used. In Figure 1, the output of frontend 22 is illustrated as being a common language file 26 containing a combination of both metadata 28 and executable instructions 30. Alternatively, the front end system 22 may generate two separate files (not shown), one for the metadata 28 and one for the

3

executable instructions 30. In alternative embodiments, the front end system 22 may only produce either metadata 28 or executable instructions 30. Within the context of this application metadata includes any information that is useful to describe executable instructions 30. By way of example only (and not limitation), such metadata may include a description of the symbols or names used in the executable instructions, information about the data types used in the executable instructions, or any other information that would be useful either by front end 22 or execution environment 32. The metadata portion 28 is produced by a metadata module 33 within the front end system 22 and the executable instructions 30 are produced by a code module 35 within the front end system 22. The front end system 22 may read or receive either metadata 28, executable instructions 30 or a combination of both, such as the common language file 26 as shown in Fig. 1.

Page 13, line 22 to page 14, line 12, please replace the existing paragraph with the following paragraph:

Additionally, in the case where the execution environment 74 is a managed runtime environment, it may provide a number of other services during program execution. As an example the system may provide a loader 75, which receives the executable file and resolves necessary references and loads the code. The environment may provide a stack walker 77, i.e., the piece of code that manages the method calls and provides for the identification of the sequence of method calls on a stack at a given point in time. A layout engine 78 may also be provided, which establishes the layout, in memory, of the various objects and other elements as part of the application to be executed. The execution environment may further provide security measures to prevent

4

unauthorized use of resources and other developer services, such as debuggers and profiling. Other types of services that can be provided by a managed execution environment include verification of code before it is executed, security facilities to determine whether certain code has permission to access certain system resources (or even execute at all), and memory management services, such as garbage collection 79.

Page 23, lines 5-17, please replace the existing paragraph with the following paragraph:

Fig 7 illustrates the operational flow for a front end portion in an alternative embodiment. In essence, the process is the same as that discussed above in conjunction with Fig. 6, except flow 300 determines at operation 304 whether a newly read statement is an import statement prior to any lexical or grammar analysis, which occurs at operation 306. That is, operation 302 is similar to operation 202 described above in that a new statement is read for analysis. Then determination step 304 determines if the statement relates to an import statement for a common library. If so, operations 308, 310, 312 and 314 are executed to load the information from the library into the symbol table. If not operations 306, 316 and 318 load the remaining portions of the source file into either the symbol table or the output stage. Additionally, operation 320 is similar to operation 220 described above and generates a common language file. Following operation 320, operational flow 33 ends at 322. The common language file can then be consumed by either other front end systems 22, 46, 56, 58 or 60 (Figs. 1, 2 and 3), or an execution environment, such as environment 74 (Fig. 3).